EFILive Command Line Reference

Paul Blackmore

EFILive Command Line Reference

First published
28 August 2016

Revised
11 June 2018

Contents

Prerequisites

Intended Audience

EFILive AutoCal resellers who need to automate and optimize the configuration and programming of AutoCal devices prior to shipping.

Computer Knowledge

It is expected that readers have a basic understanding of:

- The Windows operating system;
- Starting and using Windows applications;
- The Windows Command Line Interface;
- Windows batch files;
- Navigating folders and copying and moving files using Windows Explorer.

Lua Scripting

To make use of the scripting capabilities it is recommended to have at least a basic understanding of the Lua programming language.

https://www.lua.org/

https://www.lua.org/manual/5.3/contents.html#index

Introduction

EFILive Command Line Interface

Automation

The EFILive Command Line Interface application (CLI) provides tuners with a way to automate the following tasks using the Windows command line or Windows batch files:

- Program FlashScan/AutoCal device settings;
- Program black box scan and tune options;
- Format the FlashScan/AutoCal file systems;
- Copy tune files to FlashScan/AutoCal;
- Run Lua script files to update tune files.

Gang-Programming

As well as automation, the CLI also supports gang-programming. Gang-programming allows up to five devices to be programmed and updated simultaneously.

If you find yourself setting up and programming the same FlashScan/AutoCal configurations day after day into multiple devices, then using gang-programming will reduce your workload.

USB Requirements

When connecting multiple FlashScan/AutoCal devices to your PC, you should use an external, self-powered USB 2.0 hub. Ensure that the hub is capable of supplying the necessary power requirements. Connecting five FlashScan devices requires a self-powered hub that can provide enough current to operate itself plus five FlashScan/AutoCal devices simultaneously. Each FlashScan/AutoCal device is capable of drawing up to 250mA.

EFILive recommends using an external, self-powered, USB 2.0 hub with a power supply rated at 2A or higher.

Command Line Interface

Reference

Installation

The EFILive V8 software must be installed prior to installing or using the EFILive Command Line application.

Copy the EFILive_CmdLine.exe file into the same folder that contains the EFILive_Hapi.exe file. Usually that will be the folder:

\Program files (x86)\EFILive\V8

You may need administrator privileges to complete that operation.

Version Compatibility

Ensure that the version number of the EFILive_Hapi.exe file is V8.2.2 Build 303 or later.

To view the EFILive_Hapi.exe version number, run the EFILive Control Panel (if it is not already running) and click on the FlashScan icon in the system tray (near the clock on the Windows Task Bar), then select the [F10: About] tab page.

Command Line Syntax

```
Usage: EFILive_CmdLine [switches][ list]
   Or: EFILive_CmdLine -sName tunefile[ args]


Switches:
  -a       Display version information
           Default: Do not display version information

  -dX      Target device X where X is one of:
             AC2 = AutoCal V2
             FS2 = FlashScan V2
           Default: -dAC2

  -fX      Format target device(s) volume X where X is one of
             c = Config
             d = Data
           Default: Do not format

  -iX      Identify device on LCD where X is one of
             0 = Hide identifier
             1 = Show identifier
           Default: -i1

  -nX      Connect to exactly X devices where X is 1..5
           Default: -n1


  -oX      Set the options for the attached device(s)
           where X is one or more of:
             F  = Update firmware only if updated firmware is available
             F1 = Update firmware always
             L  = Un-link AutoCal from the attached FlashScan
             L1 = Link AutoCal to the attached FlashScan
             S  = Unset the "Can Self-Sign" option
             S1 = Set the "Can Self-Sign" option
             Vn = Set the "Max VIN Licenses" to n where n is 1..221
           Default: Do not change any options

  -q       Quiet, do not display progress/results
           Default: Not quiet

  -rX      Set remote folder to X
           Default: -r/EFILive/Config

  -sName   Run the Lua script file called Name
           Default: Do not run script file

  -vX      Set verbose detail to X where X is one of
             0: No information
             1: Normal information
             2: Verbose information
           Default: -v0

Arguments:
  list     Copy files in list[n] to device [n]
           List may be one of:
             a folder containing the files to copy
             a file containing a list of file names to copy
           Default: Do not copy any files

  tunefile attempt to open the tune file “tunefile” as file descriptor 0.
  args[n]  pass arguments to the script file in the Lua table EFI_Args.
```

Switches

Switches are used to modify the processing behavior of the CLI. If a switch is specified more than once on the command line the switch furthest to the right takes precedence.

The exception to that is the –f switch. If both the –fc and –fd switches are specified then both the Config and Data file systems are formatted.

Switches may be prefixed with either the minus symbol like this –dAC or the forward slash symbol like this /dAC.

Switches must be separated by one or more spaces and may not appear after the first non-switch argument. Any switch that does appear after the first argument will be treated as an argument.

–a

Displays version number information about the EFIive_CmdLine.exe software.

–dX (device)

The –dX switch tells the CLI which devices it will be using.

Use:

- –dAC2 if you are targeting AutoCal V2 devices.
- –dFS2 if you are targeting FlashScan V2 devices.

Targeting a combination of FlashScan and AutoCal devices is not supported.

If this switch is not supplied the default value is –dAC2.

–fX (format)

The –fX switch causes one or both of the Config and Data file systems to be formatted.

Use:

- –fc to format the Config file system.
- –fd to format the Data file system.
- –fc -fd to format both the Config and Data file systems.

No warning is displayed prior to formatting a file system. All data on the file system will be erased during the format.

It is not recommended to use the –fd switch on a FlashScan device fitted with an SD Card. SD Cards can take multiple minutes to format and the CLI application will not wait long enough for the format to complete, causing the CLI to abort.

If this switch is not supplied, then no formatting takes place.

–iX (identification)

The –iX switch shows or hides the device identification number on each devices' LCD. The device identification allows you to track which files were sent to which device.

Use:

- –i0 will hide the device identification.
- –i1 will show the device identification.

If this switch is not supplied the default value is –i1.

See Appendix A for more information on device identification.

–nX (number of devices)

The –nX switch tells the CLI how many FlashScan or AutoCal devices are connected to the USB hub. If the exact number of devices are not detected by the CLI, then no files are copied.

Use:

- –n1 for a single device.
- –n2 for two devices.
- –n3 for three devices.
- –n4 for four devices.
- –n5 for five devices.

If this switch is not supplied the default value is –n1.

–oX (options)

The –o switchis used to set or unset various options on the FlashScan or AutoCal devices.

Use:

- –oF to update the firmware if a higher firmware version file is available.
- -oF1 to update the firmware always.
- -oL to un-link AutoCal from the attached FlashScan.
- -oL1 to link AutoCal to the attached FlashScan.
- -oS to unset the "Can Self-Sign" option.
- -oS1 to set the "Can Self-Sign" option.
- -oVn to set the "Max VIN Licenses" to n where n is 1..221.

If this switch is not supplied the default is to not change any options.

–q (quiet)

The –q switch supresses output messages.

Use:

- –q to supress output messages.

If this switch is not supplied, then output messages are not supressed.

–rX (remote folder)

The –rX switch is really only useful when copying unknown file types. When known file types are copied, the files' extensions are used to infer the correct destination remote folder.

Extension	Default Remote Folder
.txt	/EFILive/Config
.obj	/EFILive/Config
.bix	/EFILive/Config
.pmm	/EFILive/Config
.dtc	/EFILive/Config
.ctz	/EFILive/Tune
.coz	/EFILive/Tune
.dat	Internal

If a file does not have one of those extensions, then you should specify the destination remote folder using this switch.

Use:

• –r/EFILive/Config	to copy files into /EFILive/Config.
• –r/EFILive/Scan	to copy files into /EFILive/Scan.
• –r/EFILive/Tune	to copy files into /EFILive/Tune.
• –r/EFILive/Tune/Read	to copy files into /EFILive/Tune/Read.

A remote folder name of /EFILive/Config will cause the file being copied to be copied into the Config file system. All other folder names cause the file to be copied to the Data file system.

An error will occur if the remote folder does not exist.

If this switch is not specified, then –r/EFILive/Config is used.

–sName

Runs the Lua script file called "Name". Additional parameters may be specified. The first parameter is the name of the default tune file which is identified in the Lua script using file descriptor 0 (zero). All other parameters are passed to the Lua script as key/value pairs in the Lua table EFI_Args. The first argument after the default tune file name has a key of 1.

See Appendix-C for more information.

–vX (verbose)

The –vX switch sets the verbosity level for diagnostic messages.

Use:

- –v0 to supress all diagnostic messages.
- –v1 to display normal diagnostic messages.
- –v2 to display all diagnostic messages.

If this switch is not supplied, then it defaults to –v0.

Using –v2 may cause a large amount of text messages to appear on the console.

Arguments

Arguments are listed on the command line after the last switch. Arguments should not begin with either of the switch prefix characters: – or /.

If you cannot avoid using an argument that begins with either of the switch prefix characters, then separate the last switch from the first argument with a single – or / character. For example if the first argument is a folder name that starts with the – character like this "–myfolder" then the command line could look like this:

```
EFILive_CmdLine -dFS2 -n2 - -myFolder1 -myFolder2
```

Notice the – between –n2 and –myFolder1? That tells the CLI that –myFolder1 is the first argument and not another switch.

Be careful not to accidentally introduce a space between the – and its switch letter like this:

```
EFILive_CmdLine -dFS2 - n2 Folder1 Folder2
```

because that will cause the "n2" to be treated as an argument instead of a switch.

You ***must*** specify as many arguments as there are devices declared by the –nX switch (or zero arguments). Each argument supplied resolves to a list of filenames that will be copied to its corresponding device.

- The first argument is the list of files that will be copied to device 1.
- The second argument is the list of files that will be copied to device 2.
- The third argument is the list of files that will be copied to device 3.

  …
  etc.

Each argument must be either:

- The name of an existing folder or;
- The name of an existing text file.

If zero arguments are specified, then no files are copied. Specifying zero arguments is useful if you just want to format the file systems or display the device ID, for example:

To format the Config file system on 3 FlashScan devices do this:

```
EFILive_CmdLine -dFS2 -n3 -fc
```

To display the device ID on 5 FlashScan devices do this:

```
EFILive_CmdLine -dFS2 -n5 -i1
```

Argument as a folder name

For non-script duties, if the argument is a folder name, then all files in that folder will be copied to the device.

Argument as a file name

For non-script duties, if the argument is a text file name, then each line in that file is the name of a file that will be copied to the device.

- Blank lines are ignored.
- Lines beginning with a semicolon are treated as comments and are ignored.
- Duplicate file names are only copied once.

Appendix A
Device Identification

It is not always possible for the EFILive software to correctly identify a set of identical USB devices by their physical connection with the PC. Especially if the PC has USB 3.0 capability. The FTDI USB drivers cannot determine the physical location ID’s of USB 3.0 ports.

Any task that requires programming a specific device with a specific set of files, should only be performed while a single device is connected.

To identify multiple devices uniquely, the EFILive Command Line Interface can request that each device displays a device ID on its LCD screen (via the –i1 switch).

When gang-programming multiple devices, it is not possible to determine which device will be assigned which id prior to connecting the devices and displaying the IDs.

Each ID that is displayed corresponds to one of the command line arguments, such that:

- files in the first argument are copied to the device displaying ID 001.
- files in the second argument are copied to the device displaying ID 002.
- files in the third argument are copied to the device displaying ID 003.
- files in the fourth argument are copied to the device displaying ID 004.
- files in the fifth argument are copied to the device displaying ID 005.

If you invoke the EFILive Command Line Interface multiple times on the same set of devices without disconnecting/reconnecting any device, then each device will retain the same ID across each invocation.

If you disconnect/reconnect one or more devices then the device ID’s of each device are not guaranteed to remain the same.

Appendix B
Integration

The CLI uses the EFILive Control Panel application (aka EFILive_Hapi.exe) to communicate with FlashScan and AutoCal devices.

When the CLI runs, it automatically starts EFILive_Hapi.exe in background (if it is not already running) and a small FlashScan icon appears in the system tray.

Normally applications that start the EFILive Control Panel register their main window with the Control Panel. The Control Panel monitors that window and when it closes, the Control Panel automatically deregisters the application. When no more applications are registered with the Control Panel, the Control Panel terminates.

The CLI does not have a window of its own that it can register with the Control Panel, so it registers the Windows Command Line window instead. That means the EFILive Control Panel will not deregister the CLI application until the Windows Command Line window is closed.

Under certain error or abnormal abort conditions the FlashScan/AutoCal devices and the EFILive Control Panel can become unresponsive. Usually because the multiplexed messages for devices have become unsynchronized.

If you see any errors with the text "…can't get control of semaphore…" or the CLI simply becomes unresponsive then you should:

- If the CLI is unresponsive, press Ctrl+C to terminate the CLI.

  Some tasks can take minutes to complete, don't confuse a long running task with the CLI hanging.

  If in doubt, use the command line switch –v2 to display diagnostic trace messages.

- Disconnect all FlashScan/AutoCal devices.
- Manually close the Control Panel.

  Right click on the FlashScan icon in the System Tray (near the clock on the Windows Task Bar) and select "Exit".

  If the Control Panel is unresponsive, use Windows Task Manager to end the EFILive_Hapi.exe process listed in the [Processes] tab page.

- Reconnect all FlashScan/AutoCal devices.
- Retry the CLI command(s).

Appendix C
Scripting

Making automated and/or repetitive changes to tune files can be accomplished using a script file. The Lua scripting language and runtime interpreter is used by the EFILive V8 software to implement scripting. More information about the Lua scripting language can be found here: https://www.lua.org/

A typical script invocation may look something like this:

```
EFILive_CmdLine –sSecure.Lua A.ctz B.ctz 1G1ABCDEFGH123456 forAutoCal
```

- The –sSecure.lua switch tells the Lua interpreter to load the Lua script file called Secure.Lua and open the file **A.ctz** as file descriptor 0.
- The argument **B.ctz** is passed to the script as argument number 1, available in the script as EFI_Args[1].
- The argument **1G1ABCDEFGH123456** is passed to the script as argument number 2, available in the script as EFI_Args[2].
- The argument **forAutoCal** is passed to the script as argument number 3, available in the script as EFI_Args[3].

Return value

The script can return a value back to the command line using the return(x) function. The value returned to the command line can be accessed from the command line or within another batch file using the pseudo environment variable %errorlevel%.

If the script is running inside the V8 software, the return value is displayed in the window where the script is running.

A typical script file that changes the VIN, sets the security mode to “cannot be viewed or modified” and saves the file (with a different name) for use with a remote AutoCal:

```
-- First check that EFILive was able to successfully initialize everything prior to starting the script.
if ( EFI_ErrNum~=tleNone ) then
  errorMessage = string.format("Can't start script, %s",EFI_ErrMsg)
  if ( EFI_CmdLine ) then
    print(errorMessage)
  else
    _efiMsgDialog(errorMessage,mtError,{mbOk})
  end
  return(EFI_ErrNum) -- return error number to command line
end

fileId = 0  -- use file specified from the command line, i.e. file descriptor 0 (zero).

-- Create table "p" which contains the file properties that we want to update.
-- Note these values may be specified on the command line if desired.
p = {
      [fpidCtrlVIN]=EFI_Args[2],     -- Set the VIN to command line argument 2.
      [fpidSecMode]=2                -- Set the Security Mode to 2="Cannot be viewed or modified",
    }                                --   requires a Master FlashScan to be connected .

r = _efiSetFileProp(fileId,p)       -- Set the properties defined in table "p".
if ( r==nil ) then
  -- if error, then print the error message on the command line.
  print(string.format("Can't set properties, %s",EFI_ErrMsg))
  return(EFI_ErrNum) -- return error number to command line
else
  -- If success, then print the new values for the two file properties.
  print(_efiGetFileProp(fileId,fpidCtrlVIN)) -- Get and print the updated VIN.
  print(_efiGetFileProp(fileId,fpidSecMode)) -- Get and print the updated security mode.
end

-- If command line argument 3 is "forAutoCal" then save this file for use with a remote AutoCal device,
--   requires a Master FlashScan to be connected.
forAutoCal = EFI_Args[3]=="forAutoCal"

-- Save the file using the filename specified as command line argument 1.
r = _efiSaveFile(fileId,EFI_Args[1],forAutoCal)
if ( r==nil ) then
  if ( EFI_ErrNum==tleNoSaveDefault ) then
    -- If file is open in the EFILive V8 software then it can't be saved by the script
    --   user must save the faile later.
    print("File has been modified, don't forget to save the file.")
  else
    print(string.format("Can't save file, %s",EFI_ErrMsg))
    return(EFI_ErrNum) -- return error number to command line
  end
else
  -I if success, then print "OK".
  print("File Saved OK.")
end
return(0) -- return 0 back to the command line to indicate success,
          --   check value on comman dline with %errorlevel%
```

To run the script, use a command line like the one shown below. Obviously substitute your own folder/file names and VIN:

```
efilive_cmdline -sC:\Users\Paul\Documents\EFILive\V8\Script\Secure.lua
C:\Users\Paul\Documents\EFILive\V8\Tune\E38\MyTune.ctz
C:\Users\Paul\Documents\EFILive\V8\Tune\E38\MyNewTune.ctz
1G1ABCDEFGH123456
forAutoCal
```

The arguments are shown on different lines for clarity only. The arguments should be specified on the same line as the command.

The output of this script when run successfully will look like this:

```
1G1ABCDEFGH123456
2
File Saved OK
```

EFILive Provided Functions

EFILive has provided functions that you can use in the Lua scripts to manipulate tune files and their settings. All EFILive provided functions start with the prefix _efi.

_efiMsgDialog(msg,type,buttons)		
Displays a message in a dialog and provides one or more buttons for the user to select.		
Returns	Button that user clicked: 0: btnOk 1: btnCancel 2: btnYes 3: btnNo 4: btnAll 5: btnNoToAll 6: btnYesToAll	
Parameters	msg	The text message to display.
	type	The dialog icon type: 0: mtInformation 1: mtConfirmation 2: mtWarning 3: mtError
	buttons	Table of one or more buttons: 0: btnOk 1: btnCancel 2: btnYes 3: btnNo 4: btnAll 5: btnNoToAll 6: btnYesToAll

_efiOpenFile(name)		
Opens a tune file (*.ctz and *.bin file formats are supported) Up to 5 files may be opened at the same time. When a script is run from the command line, the very first argument following the –sName switch is interpreted as a tune file name and automatically opened using the file descriptor 0. When a script is run from within the EFILive V8 software, the currently open tune file is assigned to file descriptor 0.		
Returns	Success: Error:	A unique integer file descriptor. nil
Parameters	name	The name of the file to open. Using a fully qualified pathname is recommended.

_efiSaveFile(fd,name,forRemote)		
Saves a ctz tune file. When a script is run from within the EFILive V8 software, the currently open tune file is assigned to file descriptor 0. In that case the file assigned to file descriptor 0 cannot be saved by the script. Once the script terminates, any modifications made by the script may be saved by the user using the V8 software.		
Returns	Success: Error:	0 nil
Parameters	fd	The unique file descriptor returned by the _efiOpenFile() function.
	Name	The name of the file to save. Using a fully qualified pathname is recommended. If the name is nil or empty the file will be saved using the file's original filename.
	forRemote	If true, then save the file for use with a remote AutoCal. A master FlashScan device must be connected to supply the license number when saving for a remote AutoCal.

_efiCloseFile(fd)		
Closes a tune file.		
Returns	Success: Error:	0 nil
Parameters	fd	The unique file descriptor returned by the _efiOpenFile() function.

_efiUpdateChecksums (fd)		
Updates the checksums in a tune file.		
Returns	Success: Error:	0 nil
Parameters	fd	The unique file descriptor returned by the _efiOpenFile() function.

_efiGetFileProp(fd,propId)		
Gets the requested file parameter value from the tune file.		
Returns	Success: The requested parameter's value. Error: nil	
Parameters	fd	The unique file descriptor returned by the _efiOpenFile() function.
	propId	The property ID of the property to be retrieved. See next table for a list of property IDs’

ID	Identifier	Description	Can be Modified
File Info			
10	fpidFileVer	Version of tune file	✗ 1
11	fpidFileName	Name of tune file	✗
12	fpidFileReadOnly	Is file marked as read-only at the Windows operating system level? (i.e. it can't be saved over)	✗
13	fpidFileModified	Has file been modified since it was opened?	✗
14	fpidFileComments	Comments	✓ 2
15	fpidFileHistory	History This property can't be modified it can only be cleared. To clear the history, set its value to an empty string.	✓ 3
16	fpidFileCreatedBy	File first saved by	✗
17	fpidFileCreatedDate	Date file first saved	✗ 4
18	fpidFileModifiedBy	File last saved by	✗
19	fpidFileModifiedDate	Date last saved	✗ 4
Controller info			
30	fpidCtrlVIN	Controller's VIN	✓ 5
31	fpidCtrlOS	Controller's OS	✗
32	fpidCtrlSerial	Controller's Serial	✗
Where file came from			
40	fpidSrcVer	Version of EFILive software that was used to read this file	✗
41	fpidSrcDevName	Device name that was used to read this file	✗
42	fpidSrcDevSer	Device serial that was used to read this file	✗
43	fpidSrcDevLic	Device license that was used to read this file	✗
44	fpidSrcDevFileSys	Device file system in use when this file was read	✗
45	fpidSrcObjVer	Version of *.obj script file	✗
46	fpidSrcObjDate	Date of *.obj script file	✗ 4
47	fpidSrcBLID	Boot loader ID string	✗
48	fpidSrcFwVer	Firmware version	✗
49	fpidSrcFwDate	Firmware date	✗ 4
50	fpidSrcBBVer	Boot Block version	✗
51	fpidSrcBBDate	Boot Block date	✗ 4
Stream Info			
60	fpidStreamID	Stream ID	✗

61	fpidStreamSubID	Stream Sub ID (i.e. E35 has E35A=0 and E35=1)	✗
62	fpidStreamName	Stream Name	✗
Remote Info			
70	fpidRemLic	Remote license - link to AutoCal	✗[6]
71	fpidRemKey	Remote user key (encrypted). Setting this value to an empty string or nil will clear all three properties (71, 72 and 73).	✓[7]
72	fpidRemFromKey	Remote user key range start (encrypted. Setting this value to an empty string or nil will clear all three properties (71, 72 and 73).	✓[7]
73	fpidRemToKey	Remote user key range end (encrypted. Setting this value to an empty string or nil will clear all three properties (71, 72 and 73).	✓[7]
74	fpidRemTryAltKey	Remote try alternative keys	✓
75	fpidRemLockFaulty	Remote assume lock may be faulty	✓
76	fpidRemVpw4x	Remote use 4x VPW	✓
77	fpidRemCumminsMode	Remote cummins fast mode 0=Normal, 1=unlimited, 2=fast	✓
Security Info			
80	fpidSecMaster	Master FlashScan license	✗[8]
81	fpidSecMode	Privacy mode: 0=open, 1=no-mod, 2=no-view	✓
82	fpidSecFlashMode	Flash mode: 0=both, 1=cal only, 2=full-only	✓
83	fpidSecAutoLock	AutoLock true/false	✓
90	fpidSecDevLic	Security device license	✓
91	fpidSecDevSer	Security device serial	✓
92	fpidSecCtrlSer	Security controller serial	✓
93	fpidSecCtrlVin	Security controller VIN	✓
94	fpidSecTargetOS	Override target operating system	✓[9]

[1] Returns 4 values, so use this type of syntax: v1,v2,v3,v4 = _getFileProp(fd, fpidFileVer)
[2] Returns a single level table with each paragraph as an entry in the table.
[3] Returns a nested table, see next section for table format.
[4] Returns 3 values, so use this type of syntax: yy,mm,dd = _getFileProp(fd, fpidFileCreatedDate)
[5] Not all controllers support VIN change.
[6] Automatically updated when “Save for Remote AutoCal” option is used.
[7] Encrypted values cannot be read, they can only be updated with new values, which are then automatically encrypted.
[8] Automatically updated when saving a file with security options active.
[9] A Pro Tuning license is required to override a target operating system.

EFILive Table Formats

When retrieving the fpidFileHistory property, the history data is returned as a nested table with the following format:

```
{
  {{year=yyyy, month=mm, day=dd, hour=hh, min=nn, sec=ss},CalID,EditVer,CalVer,Text},
  {{year=yyyy, month=mm, day=dd, hour=hh, min=nn, sec=ss},CalID,EditVer,CalVer,Text},
  {{year=yyyy, month=mm, day=dd, hour=hh, min=nn, sec=ss},CalID,EditVer,CalVer,Text},
…
}
```

Where

- yyyy is the 4 digit year
- mm is the 2 digit month (1..12)
- dd is the 2 digit day (1..31)
- hh is the 2 digit hour (00..23)
- nn is the 2 digit minute (00..59)
- ss is the 2 digit second (00..59)
- CalID is the calibration ID, if this value is blank, then the Text is the section heading.
- EditVer is the version of the software used to modify CalID
- CalVer is the version of the calibration definition file in use when CalID was modified.
- Text is either the sectionheading (if CalID is empty) or a description of the modification made to CalID (if CalID is not empty).

When retrieving the fpidFileComments property, the comments are returned as a table with the following format:

```
{
  "Comment paragraph 1",
  "Comment paragraph 2",
  "Comment paragraph 3",
…
}
```

_efiSetFileProp(fd,idTable)		
Sets all the file properties to their corresponding values from table.		
Returns	Success: Error:	The number of properties updated. nil
Parameters	fd	The unique file descriptor returned by the _efiOpenFile() function.

	idTable	The name of a Lua table that contains a list of the property values indexed by property id.
		For example to update the following properties with these specified values:
		30 (Controller's VIN) = "1G1ABCDEFGH123456"
		81 (Privacy Mode) = 2 (no-view)
		82 (Flash Mode) = 1 (cal only)
		Then a Lua table called "idTable" could be constructed like this:
		`idTable = {`
		`  [30]="1G1ABCDEFGH123456",`
		`  [81]=2,`
		`  [82]=1`
		`}`

_efiIdToStr (idType,id,prefx)		
Gets the name of a property id. Useful during debugging if/when you want to print the name of a property alongside its value.		
Returns	Success: Error:	The name of the property id. nil
Parameters	idType	One of: 0: idtMeta Lookup these identifiers. 1: idtError Lookup error number identifiers. 2: idtButton Lookup button identifiers. 3: idtDialog Lookup dialog type identifiers. 4: idtFileProp Lookup file property identifiers.
	id	Return the name of this identifier value from the set of identifiers defined by idType.
	prefix	If false then exclude the lowercase prefix of the identifier's name. For example for the identifier ***fpidFileVer***, if prefix is false then the returned name would be simply ***FileVer***.

EFILive Error Numbers

Sometimes the EFILive provided functions will fail for various reasons. Each time an EFILive function is called and fails, the two Lua variables: EFI_ErrNum and EFI_ErrMsg are set with the error number and error message respectively.

Code	Identifier	Description
$0800	tleUnknown	An unknown error occurred.
$0801	tleException	An exception occurred, the Lua variable EFI_ErrMsg contains a description of the exception.
$0802	tleIntegerExpected	Integer argument is expected but a non-integer argument was supplied.
$0803	tleIntegerOutOfRange	The integer argument was not within the expected/required range.
$0810	tleTooManyFiles	Attempt to open too many files. Limit 5.
$0811	tleInvalidFile	Attempt to use a file descriptor that is not in the range 0..4.
$0812	tleFileNotOpen	Attempt to use a file descriptor that is not open.
$0813	tleFileAlreadyOpen	Attempt to re-open an already open file.
$0814	tleFileNotIdentified	Attempt to open a file for which EFILive cannot identify the correct controller.
$0815	tleFileIsReadOnly	Attempt to update properties in a read-only file.
$0816	tleNoSaveDefault	Attempt to save the default tune file when script is running inside the V8 GUI.
$0850	tleInvalidId	Invalid id passed to _efiIdToStr() function.
$0851	tleInvalidMetaId	Invalid idType passed to _efiIdToStr() function.
$0852	tleInvalidCalId	Invalid calibration id.
$0853	tleInvalidCalPropId	Invalid calibration property id.
$0854	tleInvalidFilePropId	Invalid file property id.
$0855	tlePropReadOnly	Attempt to update a read-only file property.
$0856	tlePropNotSupported	Attempt to update a file property that is not supported for the current controller.
$0857	tlePropValueIsNil	Property value is nil when it must not be nil.
$0858	tlePropValueNotTable	Property value is not a table when a table is expected.
$0859	tlePropValueNotEmpty	Property value is not empty when an empty value is expected.
$0870	tleSecurityLocked	Attempt to modify a file's security settings when the security settings are already set.
$0871	tleAutoLockNotSupported	Attempt to set the Auto-Lock property for a file whose controller does not support Auto-Locking.
$0872	tleNoMasterFlashScan	Attempt to update/save a file property id that requires a master FlashScan when no master FlashScan is present.

$0873	tleNoProLicense	Attempt to update/save a file property id that requires the Pro Tuning license when no Pro Tuning license is present.
$0874	tleSecurityDeclined	User declined to allow security options to be saved.